

Prediction of vulnerability severity using vulnerability description with natural language processing and deep learning

Abdullahi Ahmed Abdirahman¹, Abdirahman Osman Hashi¹, Octavio Ernesto Romo Rodriguez²,
Mohamed Abdirahman Elmi¹

¹Department of Computing, Faculty member, SIMAD University, Mogadishu, Somalia

²Department of Computer Science, Faculty of Informatics, Istanbul Technical University, Istanbul, Turkey

Article Info

Article history:

Received May 19, 2023

Revised Mar 19, 2024

Accepted Apr 16, 2024

Keywords:

Convolutional neural network

Deep learning

Multi-class classification

Vulnerability severity

Word embeddings

ABSTRACT

One of the most critical aspects of a software piece is its vulnerabilities. Regardless of the years of experience, type of project, or the size of the team, it is impossible to avoid introducing vulnerabilities while developing or maintaining software. This aspect becomes crucial when the software is deployed in production or released to the final users. At that point finding vulnerabilities becomes a race between the developers and malicious intruders, whoever finds it first can either exploit it or fix it. Acknowledging this situation and using the tools and standards that we have available in the field, such as common vulnerability exposures and common vulnerability scoring systems, and based on modern researches, in this study, we propose to have an approach different from the common practices of manual classification, using a 2-layer convolutional neuronal network (CNN) to automatize the classification of vulnerabilities, speeding up this process and enabling developers to have a faster response towards vulnerabilities, producing safer software. The experimental results obtained in this study suggest that pre-trained word embeddings contributed to an increase in accuracy of approximately 2% and the overall accuracy become 0.816%.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Abdullahi Ahmed Abdirahman

Department of Computer Science, Faculty of Computing, SIMAD University

Mogadishu, Somalia

Email: aaayare@simad.edu.so

1. INTRODUCTION

Software vulnerabilities are failures in a piece of software. More explicitly, errors introduced in software systems during development or maintenance pose security risks to the host computer. They are exploitable, which means that an intruder can gain access to the system or the host computer and get information, cause damage, or even take control of the whole system. Software is always released with vulnerabilities, and regardless of that, most systems are relatively safe due to constant maintenance from the developers, who release patches and updates to solve the vulnerabilities that they find during the lifetime of the software, reducing the risk of an attacker finding and exploiting a vulnerability to cause damage [1]. This is the only approach available, since finding every error during development would make development times excessively long [2]. Having said that, one of the problems with this situation is that patches are not released to solve all of the vulnerabilities found, since it takes too long and would be detrimental for the developers. Moreover, it can cause side effects in big companies with a complex administrative structure. Furthermore, companies often lack the resources to address every vulnerability in their system since known or undiscovered flaws, defects, or weaknesses are introduced throughout the life cycle of a software-system,

resulting in security-vulnerabilities. In addition, vulnerabilities have a major impact on software-systems, resulting in billions of dollars in downtime and interruption of critical assets [3].

Meanwhile, numerous organizations and researchers have sought to evaluate and prioritize vulnerabilities based on various criteria in order to address this problem. Vendors have been ranking software vulnerabilities in their own ways without disclosing the theories and techniques that went into them. The frequency of “zero-day attacks” in recent years has highlighted the necessity of giving vulnerability fixing procedures priority. With this in mind, system administrators have to make the decision of what vulnerabilities they will address first, or if they will address them at all, and during this process, the vulnerability’s severity is the most important factor to prioritize them [4]. For instance, it is impossible to estimate the potential gains that a black-hat-hacker may get from exploiting a single vulnerability. It may hurt the organization in question permanently at the same time. The enormous sums of money that companies like Facebook spend on their bug-bounty-programs are an easy way to assess their importance. In essence, these initiatives involve hiring extra individuals in addition to the testing and security teams to look for any bugs that might endanger the system’s confidentiality, integrity, or availability. In practice, there is never enough time for updates to be deployed and vulnerabilities in software systems to be patched, which opens doors for attackers. Therefore, the development team must prioritize-the-vulnerabilities in order to handle the enormous number of them properly [5].

For that case, there are already standards in the industry aimed at addressing this problem. The most known is the common vulnerability scoring system (CVSS), which is used just for this purpose. It allows security experts to accumulate a score for vulnerabilities based on well-established metrics, using formulas designed to assign a numerical score based on the different characteristics of a vulnerability and impact metrics, and provides tools to define an order for vulnerabilities based on their severity. CVSS was designed based on expert knowledge and is meant to be used by security experts. Calculating the CVSS score is a hard task that takes a long time and requires an expert in the field. Based on that and the fact that most of the developers are not experts in security but can understand fairly enough the severity of a vulnerability based only on its description, the question to answer is: “Is it possible to accurately predict the severity level of software vulnerabilities using only their description?”

Therefore, the growing frequency of security incidents and associated issues highlights the importance for researchers, scholars, and security professionals to assess the severity of software vulnerabilities [6]. Utilizing data mining and machine learning techniques has been proposed as a crucial approach for predicting vulnerability severity (VSP) to mitigate the number of vulnerabilities and those of high severity. Analyzing textual descriptions of vulnerabilities has been employed to gauge their severity using deep learning and a simple convolutional neural network. These textual data have also been leveraged to construct a model predicting software vulnerability severity, proving the reliability and high accuracy of vulnerability descriptions for prioritizing fixes [7]. VSP models provide valuable insights into software maintenance, aiding in the timely development of application patches, assessing risks to software system operation, determining the impact of fixing and maintaining software, as well as estimating the financial implications of enhancing security. Despite significant advancements in VSP through prior research, there remains a gap in exploring how a subset of characteristics could enhance VSP effectiveness [8]. By focusing on a limited set of characteristics, the issue of dimensionality is addressed, leading to improved prediction accuracy, as evidenced in other research domains such as defect prediction, bug prediction, and text classification [9].

On the other hand, as our main contribution, this study proposes to accurately predict the severity level of software-vulnerabilities based on their descriptions using the natural language processing (NLP) approach with convolutional neuronal network (CNN). Thus, the main question is, “Is it possible to accurately predict the severity level of software vulnerabilities using only their description?”. This model will leverage description information to identify patterns and relationships between various vulnerability attributes and their associated risk levels, allowing for more efficient and accurate vulnerability management.

This paper is divided into five sections. The upcoming section explains the existing literature while discussing its limitations. In section 3, we explained the methodology and dataset used for this study. In section 4, we discuss the main findings and benchmarks, and the last section illustrates the conclusions of this study along with its future work.

2. RELATED WORK

Software vulnerabilities are weaknesses or flaws within software systems that attackers can exploit to gain unauthorized access or cause damage. These vulnerabilities can manifest across various levels of software systems, including the application layer, operating system, and network protocols. Exploring software vulnerabilities is a significant focus for both cybersecurity researchers and practitioners. Numerous studies have delved into various aspects of these vulnerabilities, such as their prevalence, characteristics, and

impact. Different strategies have been proposed to prioritize vulnerabilities, which are typically categorized into qualitative and quantitative techniques. Qualitative approaches classify vulnerabilities based on their severity into different groups, while quantitative methods assign numerical values to the severity of vulnerabilities by analyzing specific features associated with them [10], [11].

One of the most popular quantitative scoring methods is the CVSS created by US-CERT. In many organizations, it has become the standard method for determining how critical vulnerabilities in their software really are. The United States government's vulnerability management data repository, the National-vulnerability-database, uses CVSS for scoring. The degree of vulnerability may be quantitatively assessed when combined with this scoring approach. Subsequent studies [12], [13] by a number of scholars incorporated various improvements. In order to create a hybrid vulnerability prioritization technique, several studies combined quantitative and qualitative approaches. One such hybrid strategy is the vulnerability-rating and scoring method (VRSS) [14]. Potential value loss (PVL) is an additional topic of study in this field. A numerical rating scale is also available in PVL. It suggests certain indicators that might be used to determine the degree of a vulnerability [15]. Therefore, the topic of vulnerability prioritization has been studied for quite some time. However, new studies in this area have begun to use implicit features of a vulnerability in order to score or classify it. Because of the complexity and specificity of the vulnerabilities involved, this can only be done by someone with extensive knowledge in the field. The vulnerability description becomes available immediately when the testing team finds a vulnerability. The potential intensity of a vulnerability is sometimes indicated by certain traits or words. A approximate assessment of the severity degree of vulnerabilities may be supplied using these phrases in the description of the vulnerabilities. The groundwork for this study has already been laid. Singh *et al.* [16] made use of historical vulnerability data to estimate how likely a new vulnerability was to be exploited. Using the same input variables as CVSS [17], the weighted impact vulnerability scoring system assigns different values to each of those impacts. In 2015, they took it a step further by including data on the effects [18]. A method for estimating the severity of a vulnerability from a description was proposed by Le *et al.* [18]. Four methods of severity assessment were explored and assessed.

Meanwhile, machine-learning-models have previously been used to categorize vulnerabilities as high or low-risk, where vulnerabilities that have been exploited in real company networks are deemed high risk, while those that have not yet been exploited are considered low risk. Later, Russo *et al.* [19] introduced an exploit prediction scoring system (EPSS). In this study, researchers came up with the notion of associating a chance of exploitation with the disclosure of a vulnerability. Recent research works discussed vulnerability severity prediction (VSP) utilizing example data [20]. They employed machine learning and created an algorithm to choose an example collection of data to serve as training data in order to improve prediction accuracy. For instance, Allodi and Massacci [21] proposed decision trees model, which is a popular technique for classification tasks in machine learning (ML). Decision trees are constructed by recursively partitioning the data space into smaller subsets based on the most informative features. The final result is a tree-like structure that can be used to make predictions about the target variable (i.e., vulnerability risk level) for new instances.

Another popular ML technique for vulnerability classification is support vector machines (SVMs), which is a powerful classification method that works by identifying the hyperplane that best separates the data points into different classes. SVMs have been shown to be effective in classifying vulnerabilities into high- and low-risk categories based on their attributes, such as the type of vulnerability, the affected system, and the potential impact [22]. There are several studies in the academic literature that have explored the use of ML models for vulnerability classification [23]. For example, in a study by the National Institute of Standards and Technology (NIST), researchers evaluated the effectiveness of decision trees and SVMs in classifying vulnerabilities based on their severity levels. The study found that both techniques were able to accurately classify vulnerabilities with high levels of precision and recall. Another study by researchers at the University of North Carolina at Charlotte used an SVM-based approach to classify vulnerabilities based on their potential impact on critical infrastructure. The study found that the SVM model was able to achieve high levels of accuracy in predicting the potential impact of a vulnerability based on its attributes [24].

Researchers in the academic sector, however, have been striving to use machine learning and deep learning to create more precise models for exploitability prediction [25]. To better analyze vulnerabilities and forecast exploitability, several researchers are currently mining vulnerability descriptions to lessen the need for domain expertise and the temporal delay concerns inherent in CVSS. Each vulnerability has its own brief paragraph describing its nature, the products and vendors affected, a brief overview of the affected versions, the severity of the problem, the level of access an attacker needs to exploit it, and the key pieces of code that must be provided as inputs [26]. Based on their contents, it is clear that descriptions are useful for exploitability prediction. Previous research did disclose some findings, however even with them there are still the following problems. i) Ignore the particular technical words used in cybersecurity and the polysemy issue in NLP. Here are two sentences: sentence a says "A buffer overflow in lsof allows local users to obtain root

privileges”, while sentence B says, “Roses will not root in such acidic soil”. The word “root” has several meanings in phrases A and B. The acronym “Isosf” stands for 'list open files' in the world of cyber security. When describing vulnerabilities, terms like “Isosf” are essential for conveying the intended message. Previous work on exploitability prediction based on descriptions has relied on the term frequency-inverse document frequency (TF-IDF) algorithm [26], rule-based statistical approach [27], and word-embedding methods to extract semantic information from descriptions. The same representation or approach is used by these feature extraction techniques regardless of the word or phrase being processed. As a result, they fail to account for the several meanings that words might have. In addition, there are many uncommon domain-specific terms like package names, tool names, variable names, and other technical phrases in the cybersecurity domain corpus. This problem has not been addressed by any of the prior studies. ii) Ignoring the interdependencies between features retrieved from descriptions while choosing classifiers. Vulnerability descriptions are dependent texts because of their sequential nature. For example, in the phrase “root privilege”, the meaning of “root” depends on the meaning of “privilege”. For exploitability prediction, prior research has used classifiers such as SVMs, random forests (RFs) [5], naive Bayes (NBs) [26], and fully-connected neural networks (referred to as DenseNNs) [25]. These classifiers assume that characteristics may be used separately. As a result, interdependencies between descriptions are lost. iii) The community of researchers interested in exploitability prediction does not have access to a single, uniform dataset. The advancement of machine learning and deep learning algorithms relies on the availability of uniform datasets to the general audience. Researchers get data from the same open-source dataset, i.e., NVD and ExploitDB5, however there is considerable variation across studies in terms of the vulnerabilities studied, the time period studied, and the exploitability status of the vulnerabilities studied. As far as we can tell, there is no centralized dataset dedicated to exploitability prediction that is available to the general public. The fundamental reason for this is the ever-evolving nature of vulnerability databases and the fluctuating ease with which they may be exploited. Researchers are continuously on the lookout for fresh information to back up their studies. As a result, every prior research that attempts to foretell whether vulnerabilities will be exploited relies on data that the researchers themselves have gathered [26].

3. METHOD

As mentioned before, common vulnerabilities and exposures (CVE) and CVSS are standards in the industry, CVE is a repository of vulnerabilities and CVSS provides a scoring system for them. A combination of these two standards is found on the [6], from which the data used in this work was taken. CVE consists of many attributes such as ID, vendor, and vulnerability type. but most of them will be ignored for this work, keeping only the vulnerability description. CVSS provides a score in a scale of 0 to 10 with one decimal point, however, this score will be transformed to for classes for classification purposes. Previous research has been done in the subject and also used the same standard which is going to be our benchmark. The target of this research is the same, trying to predict vulnerability severity using CNN. The key aspects of our research framework are:

3.1. Dataset

As mentioned previously, the data was gathered from an online repository containing a combination of CVE and CVSS data. This data had to go through several processes of cleaning previous analysis. Specifically, the original data contained various issues that had to be addressed. Firstly, it included unnecessary data that did not contribute to the analysis. This included metadata such as timestamps and sources, which were removed to focus only on the relevant information. Secondly, the data contained inconsistencies in punctuation and letter case, which had to be standardized. This was done to reduce the variability of the data and enable the models to learn from consistent patterns. Thirdly, the data contained redundant information in the form of duplicated descriptions. These were identified and removed to prevent bias in the analysis. Fourthly, the descriptions of the vulnerabilities in the data were sometimes too short and lacked sufficient detail. In such cases, additional information was gathered from external sources to augment the data and ensure its relevance. Finally, the data contained non-standard information such as tokens, directories, URLs, and other irrelevant information that was removed to enable the machine learning models to focus only on the relevant text data. After the first stage of the cleaning process, the cleaned data has the format <vulnerability description, expert rated CVSS severity level> as the upcoming Figure 1 illustrates.

3.2. Data pre-processing

In this study, the data pre-processing step played a pivotal role in the overall data analysis pipeline, and it was carried out utilizing the versatile and widely adopted Python programming language. This allowed for efficient and flexible data manipulation, transformation, and preparation, ensuring that the data was in an

appropriate format for subsequent analysis using machine learning models. The pre-processing step encompassed a series of essential procedures aimed at enhancing the quality and reliability of the data. One crucial aspect was data cleaning, which involved identifying and addressing any inconsistencies, errors, or missing values within the dataset which we have done it. Through techniques like outlier detection, imputation, and data validation, we were able to rectify inaccuracies and ensure the integrity of the data before further analysis. Another integral part of the pre-processing step was data formatting, which involved standardizing the structure, representation, and encoding of the data. This process was crucial for ensuring compatibility and consistency across different data sources or features. Techniques such as normalization, scaling, and encoding were employed to transform the data into a consistent and unified format, thereby enabling fair comparisons and meaningful analysis.

The resulting dataset used in this study contained 11,669 descriptions, which were divided into four classes as Figure 2 demonstrates and can be seen clearly by score class and number of reports. Of the reports, nearly half were of medium severity, while the rest were classified as low, high, or critical severity. The data was structured in rows, with each row containing a description of a vulnerability and its corresponding severity level. Although the data had undergone some initial cleaning in the data gathering and preparation stage, some punctuation marks, special characters, and other artifacts were still present in the text as Figure 3 shows as previewing descriptions. Therefore, additional text pre-processing techniques such as tokenization and stop word removal were applied to the data to extract relevant features and reduce the noise in the data which will discuss it next section of word embedding. However, the pre-processed dataset was then divided into two sets: a training set comprising 70% of the data and a testing set comprising the remaining 30%. The training set was used to train the machine learning models, while the testing set was used to evaluate their performance.

```
456 In Metinfo 7.0.0beta,medium
457 In the Linux kernel before 5.0.3,high
458 In the Linux kernel before 5.0,high
459 In phpBB before 3.1.7-PL1,medium
```

Figure 1. Vulnerability description

```
medium      5780
low         2060
high        1954
critical    1876
Name: score, dtype: int64
```

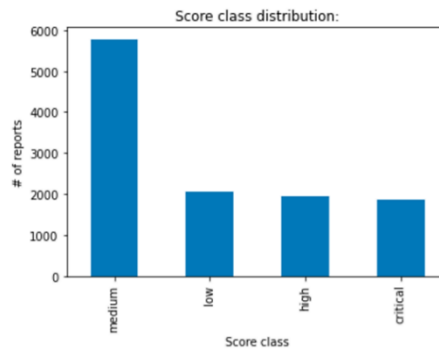


Figure 2. Overview of class disruptions

```
description
0 invenio-previewer before 1.0.0a12 allows XSS
1 Discourse before 2.3.0 and 2.4.x before 2.4.0...
2 Discourse before 2.3.0 and 2.4.x before 2.4.0...
3 ASH-AIO before 2.0.0.3 allows an open redirect
4 graphql-engine (aka Hasura GraphQL Engine) bef...
...
11665 Use-after-free vulnerability in the nfnql_zcop...
11666 Unspecified vulnerability in the PeopleSoft En...
11667 Unspecified vulnerability in the Core RDBMS co...
11668 Unspecified vulnerability in the Oracle Agile ...
11669 Unspecified vulnerability Oracle the MySQL Ser...
```

Figure 3. Previewing descriptions

3.3. Word embedding

This study incorporates the utilization of three distinct word embedding models, a fundamental technique in natural language processing, to represent the input text data within a high-dimensional vector space. By employing word embeddings, the study aims to capture the nuanced semantic relationships that exist between words, thus facilitating the optimal performance of machine learning models in subsequent tasks. Word embeddings play a crucial role in bridging the gap between human language and machine learning algorithms. These models generate dense vector representations, where words with similar meanings or contextual usage are located closer to one another in the embedding space. This enables machine learning models to leverage the inherent semantic associations and contextual information encoded within the embeddings, enhancing their ability to understand and analyze textual data effectively. By employing three different word embedding models, the study aims to explore the strengths and weaknesses of each approach. This comparison provides valuable insights into the performance and suitability of different embedding techniques for specific applications or domains. Furthermore, utilizing multiple embeddings helps to capture diverse perspectives and nuances in the underlying textual data, enabling a more comprehensive representation of the semantic relationships between words. The use of word embeddings in this study is particularly advantageous as it overcomes the limitations of traditional approaches that rely solely on numerical or binary representations of words. By embedding words in a high-dimensional vector space, the study leverages the full power of semantic relationships, enabling machine learning models to perform more accurately in tasks such as text classification, sentiment analysis, or information retrieval.

The first word embedding model employed in this study is based on the CVE vulnerability description corpus, which contains 3,003,787 tokens and has a vocabulary size of 64,184. The word embedding model derived from this corpus is 300-dimensional, meaning each word is represented as a vector in a 300-dimensional space. This model is specifically trained on the domain of vulnerability descriptions, which are often technical and contain specific jargon related to cybersecurity. The second word embedding model utilized in this study is based on the stack overflow corpus, which comprises 9,646,800,250 tokens and has a vocabulary size of 800,818. The word embedding model derived from this corpus is also 300-dimensional and is trained on a diverse range of technical topics related to software development and information technology. The use of this model allows the machine learning models to capture domain-specific nuances and patterns that may be missed by a more general model. The third and final word embedding model utilized in this study is based on the GoogleNews-vectors, which is a pre-trained model on a large general text corpus. This model contains three million 300-dimensional English word vectors and is not specific to any particular domain or topic. The use of this model enables the machine learning models to capture general semantic relationships between words and can be useful in cases where the input text data is not domain specific.

3.4. Tokenization

The purpose of this process was to enhance the system's performance by incorporating word embeddings. To achieve this, the descriptions underwent a transformation from strings into an array of words. The process involved several steps, outlined as follows: Firstly, the raw sentence was converted into an array of words. This step enabled the system to treat each word as a separate entity, facilitating further analysis and manipulation. Secondly, various elements such as punctuation marks, numbers, paths, and other extraneous characters were removed from the array of words. This cleansing process eliminated potential noise or irrelevant information that could hinder the accurate representation of the text. Lastly, all the words in the array were converted to lowercase. This normalization step ensured that the system treated words regardless of their original capitalization, avoiding duplication or misinterpretation due to inconsistent casing. By performing these steps, the descriptions were transformed into a preprocessed format suitable for word embedding techniques. This allowed the system to capture the semantic relationships and contextual information encoded within the words, ultimately enhancing its ability to analyze and understand the text accurately.

Figure 4 provides a visual representation of the token implementation in the context of the study. The figure demonstrates that each word within the text is represented as an individual element in array form. This tokenization process facilitates the identification and isolation of each word, enabling subsequent steps such as stemming and lemmatization to be applied effectively. By representing each word as an array element, the token implementation allows for straightforward access and manipulation of the text. This becomes particularly advantageous when implementing stemming and lemmatization techniques, as these processes often rely on identifying the base or root form of words. With the individual words organized in array format, the system can easily locate and process each word independently. Stemming, a technique used to reduce words to their root or base form, often involves removing prefixes or suffixes to achieve word normalization. The token implementation shown in Figure 4 enables the system to identify and modify each

word individually, simplifying the application of stemming algorithms. Similarly, lemmatization, which aims to transform words to their dictionary or base form considering their meaning and context, can benefit from the token implementation. The array representation allows for efficient identification and retrieval of the base form of words, as needed for accurate lemmatization.

```

0      [invenio, previewer, before, allows, xss]
1      [discourse, before, and, before, beta, lacks, ...
2      [discourse, before, and, before, beta, lacks, ...
3      [ash, aio, before, allows, an, open, redirect]
4      [graphql, engine, aka, hasura, graphql, engine...
```

Figure 4. Token implementation

3.5. Stemming and lemmatization

During the subsequent step of the process, the tokens derived from the previous step in Figure 4 were subjected to further transformation in order to obtain their root words. This transformation was achieved through the implementation of stemming, lemmatization, or no modification at all. The outcomes of these three independent experiments were captured and recorded in the dataset, as illustrated in Figure 5. Figure 6 provides a visual representation of the impact of stemming on the dataset. After applying stemming, words were reduced to their base or root form by removing prefixes or suffixes. This process aimed to group together different variations or inflections of the same word, allowing for a more compact and unified representation of the vocabulary. By implementing stemming, the system effectively consolidated words with similar meanings or semantic relationships, facilitating subsequent analysis or modeling tasks. The resulting reduction in word variations reduced the dimensionality of the dataset, potentially improving computational efficiency and reducing the potential impact of noise or spurious variations within the text data. Stemming, as depicted in Figure 6, plays a crucial role in simplifying the representation of words, particularly in tasks such as information retrieval, text mining, or sentiment analysis. It provides a streamlined and consistent view of the words, allowing for more efficient analysis, search, or classification.

```

0      [invenio, previewer, before, allows, xss]
1      [discourse, before, and, before, beta, lacks, ...
2      [discourse, before, and, before, beta, lacks, ...
3      [ash, aio, before, allows, an, open, redirect]
4      [graphql, engine, aka, hasura, graphql, engine...
```

Figure 5. Token implementation

```

0      [invenio, preview, befor, allow, xss]
1      [discours, befor, and, befor, beta, lack, conf...
2      [discours, befor, and, befor, beta, lack, conf...
3      [ash, aio, befor, allow, an, open, redirect]
4      [graphql, engin, aka, hasura, graphql, engin, ...
5      [docker, credenti, helper, befor, ha, doubl, f...
6      [pars, server, befor, allow, account, enumer]
7      [pars, server, befor, allow, do, after, ani, p...
8      [smokedetector, intention, doe, automat, deplo...
9      [misskei, befor, allow, hijack, user, token]
```

Figure 6. Stemming implementation

On the other hand, after applying lemmatization, a linguistic technique used in natural language processing, it can be observed that it effectively reduces words to their base form, taking into consideration the semantic meaning and contextual relevance of each word. This process is particularly useful in tasks such as information retrieval, text mining, and machine translation. When employing lemmatization, a variety of linguistic resources, such as lexical knowledge and morphological analysis, are leveraged to convert words to their dictionary form. By doing so, lemmatization aims to capture the essence and core meaning of words, ensuring a more accurate representation of the underlying concepts being conveyed. Figure 7 depicts the application of lemmatization, highlighting the transformation of words within a given dataset. By examining the figure, it becomes evident that lemmatization not only simplifies the word forms but also captures the

intended sense of each word within its respective context. This aspect of lemmatization distinguishes it from other word normalization techniques, such as stemming, which often produce truncated or incomplete word forms.

```

0      [invenio, previewer, before, allows, x]
1  [discourse, before, and, before, beta, lack, c...
2  [discourse, before, and, before, beta, lack, c...
3      [ash, aio, before, allows, an, open, redirect]
4  [graphql, engine, aka, hasura, graphql, engine...
5  [docker, credential, helper, before, ha, doubl...
6  [parse, server, before, allows, account, enum...
7  [parse, server, before, allows, do, after, any...
8  [smokedetector, intentionally, doe, automatic,...
9  [misskey, before, allows, hijacking, user, token]

```

Figure 7. Lemmatization implementation

Several important characteristics should be mentioned regarding this step: firstly, independent experiments were conducted using both stemming and lemmatization techniques. These experiments aimed to explore the impact of each approach on the text data and evaluate their effectiveness in achieving the desired outcomes. In the case of stemming, the Porter Stemmer algorithm was employed. Porter Stemmer is a widely used stemming algorithm that applies a set of rules to reduce words to their base or root form by removing common prefixes and suffixes. Its application in this step helped simplify the vocabulary by grouping together words with similar roots. On the other hand, for lemmatization, the WordNetLemmatizer was utilized. The WordNetLemmatizer is a lemmatization tool that takes into consideration both the word's morphology and its context. It aims to convert words to their dictionary or base form, thereby preserving the semantic meaning and enhancing the accuracy of subsequent analysis.

It is worth noting that despite the advantages of stemming and lemmatization, one potential limitation is the possibility of losing some key words during the process. Stemming, in particular, can result in the removal of prefixes or suffixes, which may alter the meaning of certain words or lead to the loss of specific contextual information. Similarly, lemmatization relies on predefined rules and linguistic knowledge, which may not cover all domain-specific terms or rare word forms. Therefore, while stemming and lemmatization offer valuable benefits in simplifying and standardizing the text data, it is essential to be mindful of the potential trade-offs and the possibility of missing some important keywords. The choice between stemming, lemmatization, or neither depends on the specific requirements of the task at hand and the nature of the text data being analyzed.

3.6. Convolutional neural network

The convolutional neural network (CNN) developed for this study is a two-layer CNN designed to classify vulnerability reports based on their severity. The input to CNN is the tokenized version of the report description obtained through the tokenization process. The CNN was implemented using PyTorch and the Adam optimizer was used to optimize the CNN's parameters.

During the training process, CNN was trained to minimize the cross-entropy loss function, which is commonly used for multi-class classification problems. The softmax activation function was used as the final layer of the CNN to produce a probability distribution over the four possible severity classes. CNN was trained for a total of five epochs, meaning that the entire training dataset was used five times to update the CNN's parameters. This number of epochs was chosen based on empirical observations of CNN's training progress and its ability to converge to a satisfactory solution.

4. RESULTS AND DISCUSSION

As previously highlighted, a diverse range of experiments were meticulously conducted with the aim of attaining optimal performance. To comprehensively explore the potential of the system, a total of six independent experiments were carried out, each involving distinct input configurations for the CNN and word embeddings. Table 1 provides a concise summary of the different combinations utilized in these experiments. The table presents a clear overview of the specific input variations employed, allowing for easy comparison and analysis of the experimental outcomes. These combinations were carefully designed to test different configurations and evaluate their impact on the performance of the system. By systematically varying the input parameters, such as the choice of word embeddings and the configuration of the CNN, the

researchers aimed to identify the most effective combination for their specific task. This approach enabled them to explore the influence of different embedding models, hyperparameters, and network architectures on the overall system performance.

Table 1. Word embedding

| | Tokenized description | Stemmed tokens | Lemmatized tokens |
|--|-----------------------|----------------|-------------------|
| Trained with descriptions corpus | x | x | x |
| Pre-trained with stack overflow corpus | x | x | x |

4.1. Results

The numerical results of CNN's performance were obtained by assigning a numerical label to each of the four severity classes. Specifically, the classes were represented numerically as follows: critical was assigned the numerical value of 3, high was assigned the numerical value of 2, medium was assigned the numerical value of 1, and low was assigned the numerical value of 0. This numerical representation of the severity classes allowed for the calculation of performance metrics such as precision, recall, and F1 score. These metrics were used to evaluate the performance of the CNN on the testing dataset, which was held out from the training process. In general, the numerical results obtained from the testing dataset indicated that the CNN achieved high accuracy and F1 score, suggesting that it was effective in classifying vulnerability reports according to their severity. The upcoming figures illustrates the results achieved in term of steam, lemm and tokenized.

Figure 8 shows the result of the steamed token only, and it's the one that has achieved the lowest accuracy, scoring 0.66% compared to the lemmatized token and tokenized alone. Meanwhile, Figure 9 illustrates lemmatized alone as achieving average accuracy compared to steam and tokenized alone, which is 0.67%. However, tokenized alone scored the highest accuracy, which is 0.68%, as shown in Figure 10 compared to other accuracy.

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.81 | 0.50 | 0.62 | 617 |
| 1 | 0.64 | 0.91 | 0.75 | 1743 |
| 2 | 0.74 | 0.13 | 0.23 | 604 |
| 3 | 0.64 | 0.63 | 0.64 | 537 |
| accuracy | | | 0.66 | 3501 |
| macro avg | 0.71 | 0.54 | 0.56 | 3501 |
| weighted avg | 0.69 | 0.66 | 0.62 | 3501 |

Index({'iter', 'loss'}, dtype='object')

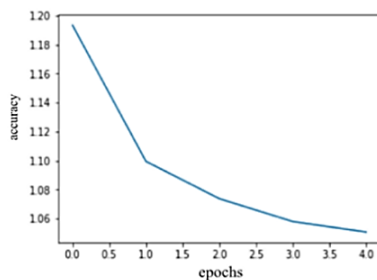


Figure 8. Stemmed token

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.77 | 0.47 | 0.59 | 617 |
| 1 | 0.64 | 0.90 | 0.75 | 1743 |
| 2 | 0.81 | 0.16 | 0.27 | 604 |
| 3 | 0.68 | 0.69 | 0.68 | 537 |
| accuracy | | | 0.67 | 3501 |
| macro avg | 0.73 | 0.56 | 0.57 | 3501 |
| weighted avg | 0.70 | 0.67 | 0.63 | 3501 |

Index({'iter', 'loss'}, dtype='object')

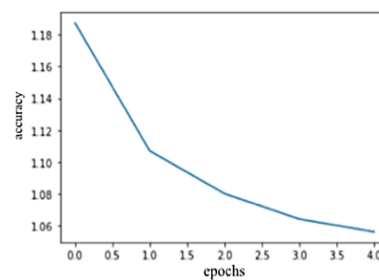


Figure 9. Lemmatized token

4.2. Discussions

The experimental results obtained in this study suggest that pre-trained word embeddings contributed to an increase in accuracy of approximately 2%. Moreover, it was found that stemming or lemmatizing tokens had an impact on the accuracy of the system. Specifically, the accuracy varied depending on the type of stemming or lemmatizing algorithm used. Increasing the number of epochs from 5 to 10 led to a modest improvement of approximately 1% in accuracy; however, this came at the cost of a significant increase in execution time. Notably, running the system with 5 epochs took approximately 30 to 40 minutes, while using 10 epochs required around 2 hours. Finally, it was found that using 80% of the available data for training resulted in an accuracy increase of approximately 2%. The accuracy result is comparable to the previous research, and also to its baseline models as Table 2.

| | precision | recall | f1-score | support |
|---------------------|-------------|-------------|-------------|-------------|
| 0 | 0.83 | 0.48 | 0.61 | 617 |
| 1 | 0.65 | 0.91 | 0.76 | 1743 |
| 2 | 0.75 | 0.20 | 0.32 | 604 |
| 3 | 0.72 | 0.69 | 0.70 | 537 |
| accuracy | | | 0.68 | 3501 |
| macro avg | 0.73 | 0.57 | 0.60 | 3501 |
| weighted avg | 0.71 | 0.68 | 0.65 | 3501 |

Index(['iter', 'loss'], dtype='object')

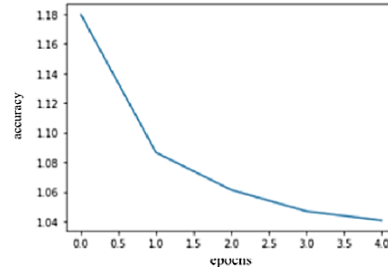


Figure 10. Tokenized description

The main factors that influenced the results of the study as can be seen from Table 2 is that firstly, the number of epochs used in the study by the previous authors was 150, whereas in this study, we used only 5 and 10 epochs. This difference in the number of epochs could have led to differences in accuracy levels. Secondly, the dataset used in the previous study consisted of 23,732 rows, whereas in our study, we used a smaller dataset of 11,669 rows. This difference in dataset size could have affected the performance of CNN. Finally, the previous study used a different optimization technique, where they used 10% of the data for optimizing hyperparameters in the CNN, whereas we used Adam optimizer. The choice of optimization technique can significantly affect the performance of tCNN.

Table 2. Accuracy of our approach and the baseline methods

| | | Accuracy |
|--------------|--------------------------------|----------|
| Baseline1 | TFID + SVM | 0.552 |
| Baseline2 | Word embedding + SVM | 0.583 |
| Baseline3 | Word embedding + 2-layer CNN | 0.775 |
| Baseline4 | Word embedding + CNN with LSTM | 0.772 |
| Our Approach | Word embedding + 1-layer CNN | 0.816 |

5. CONCLUSION

In this study, a deep learning algorithm was developed to estimate the CVSS score of a vulnerability based solely on its CVE text description. Multiple prototypes and setups were evaluated, and it was found that a multi-task architecture provided the best results. This approach makes sense since predicting several measures simultaneously can improve the model's performance when training weights are pooled. The proposed method simplifies vulnerability ranking for non-security experts by using the publicly available description when a vulnerability is disclosed. This allows system administrators to quickly assess the severity of a vulnerability, the ease of its exploitation, and the potential damage it could cause to affected systems if exploited.

To improve the model's performance, hyperparameters such as the loss weights may be further experimented with in future studies. Furthermore, larger models such as RoBERTa may be explored to provide more accurate predictions with less overall error. Additionally, the potential use of this method in estimating issue severity will be investigated, and the model will be integrated into a standalone application with a user interface. This application will automatically compute the CVSS score of newly reported vulnerabilities from the Twitter CVE account and notify users of potential risks.




REFERENCES

- [1] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 125–136, doi: 10.1109/ICSME.2017.52.




- [2] X. Gong, Z. Xing, X. Li, Z. Feng, and Z. Han, "Joint prediction of multiple vulnerability characteristics through multi-task learning," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, Nov. 2019, pp. 31–40, doi: 10.1109/ICECCS.2019.00011.
- [3] R. Sibal, R. Sharma, and S. Sabharwal, "Prioritizing software vulnerability types using multi-criteria decision-making techniques," *Life Cycle Reliability and Safety Engineering*, vol. 6, no. 1, pp. 57–67, Mar. 2017, doi: 10.1007/s41872-017-0006-8.
- [4] X. Ni, J. Zheng, Y. Guo, X. Jin, and L. Li, "Predicting severity of software vulnerability based on BERT-CNN," in *2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI)*, Jul. 2022, pp. 711–715, doi: 10.1109/ICCEAI5464.2022.00151.
- [5] P. Wang, Y. Zhou, B. Sun, and W. Zhang, "Intelligent prediction of vulnerability severity level based on text mining and XGBboost," in *11th International Conference on Advanced Computational Intelligence, ICACI 2019*, Jun. 2019, pp. 72–77, doi: 10.1109/ICACI.2019.8778469.
- [6] F. Shi, S. Kai, J. Zheng, and Y. Zhong, "XLNet-based prediction model for CVSS metric values," *Applied Sciences*, vol. 12, no. 18, Sep. 2022, doi: 10.3390/app12188983.
- [7] M. J. Tang, J. Yin, M. Alazab, J. Cao, and Y. Luo, "Modeling of extreme vulnerability disclosure in smart city industrial environments," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 6, pp. 4150–4158, 2021, doi: 10.1109/TII.2020.3022182.
- [8] I. Babalau, D. Corlatescu, O. Grigorescu, C. Sandescu, and M. Dascalu, "Severity prediction of software vulnerabilities based on their text description," in *Proceedings - 2021 23rd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2021*, Dec. 2021, pp. 171–177, doi: 10.1109/SYNASC54541.2021.00037.
- [9] G. Spanos and L. Angelis, "Impact metrics of security vulnerabilities: analysis and weighing," *Information Security Journal*, vol. 24, no. 1–3, pp. 57–71, Jun. 2015, doi: 10.1080/19393555.2015.1051675.
- [10] A. Younis, Y. K. Malaiya, and I. Ray, "Assessing vulnerability exploitability risk using software properties," *Software Quality Journal*, vol. 24, no. 1, pp. 159–202, Mar. 2016, doi: 10.1007/s11219-015-9274-6.
- [11] M. Anjum, P. K. Kapur, V. Agarwal, and S. K. Khatri, "Assessment of software vulnerabilities using best-worst method and two-way analysis," *International Journal of Mathematical, Engineering and Management Sciences*, vol. 5, no. 2, pp. 328–342, Apr. 2020, doi: 10.33889/IJMEMS.2020.5.2.027.
- [12] L. Alodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *ACM Transactions on Information and System Security*, vol. 17, no. 1, pp. 1–20, Aug. 2014, doi: 10.1145/2630069.
- [13] A. K. Shrivastava and R. Sharma, "Modeling vulnerability discovery and patching with fixing lag," in *Communications in Computer and Information Science*, vol. 956, Springer Singapore, 2019, pp. 569–578.
- [14] Y. Jiang and Y. Atif, "Towards automatic discovery and assessment of vulnerability severity in cyber-physical systems," *Array*, vol. 15, Sep. 2022, doi: 10.1016/j.array.2022.100209.
- [15] A. Okutan and M. Mirakhorli, "Predicting the severity and exploitability of vulnerability reports using convolutional neural nets," in *Proceedings of the 3rd International Workshop on Engineering and Cybersecurity of Critical Systems*, May 2022, pp. 1–8, doi: 10.1145/3524489.3527298.
- [16] U. K. Singh, C. Joshi, and D. Kanellopoulos, "A framework for zero-day vulnerabilities detection and prioritization," *Journal of Information Security and Applications*, vol. 46, pp. 164–172, Jun. 2019, doi: 10.1016/j.jisa.2019.03.011.
- [17] G. Spanos, A. Sioziou, and L. Angelis, "WIVSS: a new methodology for scoring information systems vulnerabilities," in *ACM International Conference Proceeding Series*, Sep. 2013, pp. 83–90, doi: 10.1145/2491845.2491871.
- [18] T. H. M. Le, H. Chen, and M. A. Babar, "A survey on data-driven software vulnerability assessment and prioritization," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–39, Dec. 2022, doi: 10.1145/3529757.
- [19] E. R. Russo, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities," *Journal of Systems and Software*, vol. 156, pp. 84–99, Oct. 2019, doi: 10.1016/j.jss.2019.06.001.
- [20] A. Khazaei, M. Ghasemzadeh, and V. Derhami, "An automatic method for CVSS score prediction using vulnerabilities description," *Journal of Intelligent and Fuzzy Systems*, vol. 30, no. 1, pp. 89–96, Aug. 2016, doi: 10.3233/IFS-151733.
- [21] L. Alodi and F. Massacci, "Security events and vulnerability data for cybersecurity risk estimation," *Risk Analysis*, vol. 37, no. 8, pp. 1606–1627, Aug. 2017, doi: 10.1111/risa.12864.
- [22] R. Malhotra and Vidushi, "Severity prediction of software vulnerabilities using textual data," in *Advances in Intelligent Systems and Computing*, vol. 1245, Springer Singapore, 2021, pp. 453–464.
- [23] N. Bhatt, A. Anand, and V. S. S. Yadavalli, "Exploitability prediction of software vulnerabilities," *Quality and Reliability Engineering International*, vol. 37, no. 2, pp. 648–663, Sep. 2021, doi: 10.1002/qre.2754.
- [24] P. K. Kudjo, J. Chen, S. Mensah, R. Amankwah, and C. Kudjo, "The effect of Bellwether analysis on software vulnerability severity prediction models," *Software Quality Journal*, vol. 28, no. 4, pp. 1413–1446, 2020, doi: 10.1007/s11219-019-09490-1.
- [25] G. Spanos and L. Angelis, "A multi-target approach to estimate software vulnerability characteristics and severity scores," *Journal of Systems and Software*, vol. 146, pp. 152–166, Dec. 2018, doi: 10.1016/j.jss.2018.09.039.
- [26] R. Scandariato, J. Walden, A. Hovsepian, and W. Joosen, "Predicting vulnerable software components via text mining," *IEEE Transactions on Software Engineering*, vol. 40, no. 10, pp. 993–1006, Oct. 2014, doi: 10.1109/TSE.2014.2340398.
- [27] M. S. Hoque, N. Jamil, N. Amin, and K. Y. Lam, "An improved vulnerability exploitation prediction model with novel cost function and custom trained word vector embedding," *Sensors*, vol. 21, no. 12, Jun. 2021, doi: 10.3390/s21124220.

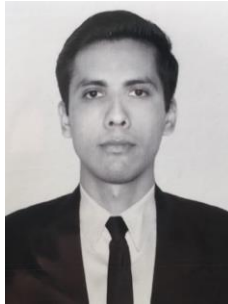
BIOGRAPHIES OF AUTHORS






Abdullahi Ahmed Abdirahman    is a lecturer at Faculty of Computing, SIMAD University. He has a master of computer science from Universiti Putra Malaysia (UPM) and bachelor of information technology from SIMAD University. He is a junior researcher. His research areas are information system and database management system. He can be contacted at email: aaayare@simad.edu.so.






Abdirahman Osman Hashi    is a former lecturer in computer science at the Faculty of Computing, SIMAD University. He holds a master's degree from the University of Technology Malaysia (UTM) and a bachelor of computer science from SIMAD University, and he is a regular PhD candidate at Istanbul Teknik University. His research areas are deep learning and artificial intelligence. He is a computer scientist with diverse working experience, including programming and software development. He has extensive knowledge of software development and artificial intelligence. He can be contacted at email: hashi1@simad.edu.so.



Octavio Ernesto Romo Rodriguez    is a student graduated with honors from Guadalajara University, currently doing a master of computer science in Istanbul Teknik University. His area of research is drone swarm technology and artificial intelligence. He has extensive experience in the industry as a software developer and server management, which he applies in the field in his research. He can be contacted at email: rodriguez19@itu.edu.tr.



Mohamed Abdirahman Elmi    is a lecturer in the Faculty of Computing at SIMAD University, teaches courses in electronic commerce, project management, relational database management systems and management information system. He holds master degree from Open University Malaysia (OUM) and bachelor of information technology from SIMAD University. His research areas are information technology project management, electronic commerce and business information systems. He can be contacted at email: m.abdirahman@simad.edu.so.